De la KAM avec un Processus d'Ordre Supérieur

D. Pous[1] & A. Schmitt[2]

1 : CNRS, damien.pous@ens-lyon.fr
2 : Inria, alan.schmitt@inria.fr

Résumé

Nous présentons un encodage simple et direct de la machine abstraite de Krivine (KAM) dans le calcul de processus d'ordre supérieur HOcore, en utilisant un nombre très restreint de canaux de communication. Cet encodage montre qu'il est possible de capturer l'expressivité du λ-calcul en HOcore dès que l'on fixe l'ordre d'évaluation. Nous donnons également une nouvelle borne inférieure pour le nombre minimal de restrictions nécessaire pour rendre l'équivalence de programmes dans HOcore indécidable. [1]

1. Introduction

Le calcul de processus HOcore est remarquable par sa similarité au λ-calcul, auquel il ajoute une notion de concurrence. Malgré cette similarité, aucun encodage du λ-calcul en HOcore n'a été présenté à ce jour. En effet, l'appariement entre une fonction et son argument est très syntaxique et très rigide en λ-calcul, alors qu'il est beaucoup plus lâche en HOcore car il correspond à la présence simultanée sur le même nom de canal d'un message et d'un récepteur pour ce message. Cette différence cruciale, couplée à l'impossibilité de générer de nouveaux noms de canaux, implique que toute traduction doit fixer le nombre de redex pouvant être activés en parallèle. Ce nombre pouvant être non-borné pour certains λ-termes, ceci empêche toute traduction *tant que la stratégie d'évaluation n'a pas été fixée.* C'est cette deuxième voie que nous explorons ici, en choisissant une stratégie en appel par nom, déclinée sous la forme d'une machine abstraite de Krivine (KAM).

Une fois ce choix de conception arrêté, la traduction est très naturelle : on représente la structure récursive de la pile de la KAM comme deux messages transportant respectivement le terme de tête et, récursivement, la queue de la pile. La β-réduction est simplement la communication entre le terme actif, représentant la fonction, avec la tête de la pile. De manière surprenante, cette traduction permet également de capturer l'opérateur de contrôle call-cc de la KAM. La réification de la continuation en tant que pile contenue dans un message permet en effet de facilement la dupliquer ou la remplacer.

Les leçons que l'on peut tirer de cet encodage sont doubles. Tout d'abord, en ce qui concerne l'expressivité de HOcore, il montre comment on peut s'appuyer sur l'ordre supérieur pour atteindre un calcul Turing complet. Les travaux précédents [5] étudiaient l'expressivité en se basant sur des machines de Minsky, qui n'ont besoin que de savoir compter et de détecter qu'un compteur vaut 0. L'ordre supérieur n'est pas nécessaire pour compter, il est en revanche utilisé pour détecter l'égalité à 0 (voir [3] et [1] pour des versions de calculs sans ordre supérieur utilisant d'autres fonctionnalités pour détecter le 0). La seconde leçon porte sur la décidabilité de la congruence barbue. En effet, nous avons montré précédemment que la congruence barbue est décidable pour HOcore [5, 2] si aucun nom de canal n'est caché, et cette congruence devient indécidable si quatre noms de canaux sont cachés. Notre traduction n'ayant besoin que de deux noms de canaux, il permet d'affiner le nombre minimal

1. Ce travail a bénéficié du soutien de l'Agence Nationale pour la Recherche dans le cadre du projet PiCoq ANR 10 BLAN 0305.

de restrictions globales nécessaire pour rendre la congruence barbue forte indécidable : il en suffit de deux.

Le reste de ce papier est organisé comme suit. Nous présentons le calcul HOcore en section 2, et la KAM en section 3. La traduction et sa preuve de correspondance opérationnelle est présentée en section 4 avant de conclure en section 5.

2. Présentation de HOcore

2.1. Syntaxe

Le calcul HOcore [5] peut être vu comme une restriction du π-calcul d'ordre supérieur [7], auquel on aurait enlevé l'opérateur de restriction de nom. Il peut également être vu comme un λ-calcul parallèle, où l'application d'une fonction $\lambda x.Q$ à un argument P est remplacée par une communication sur un canal a entre un envoi de message $\overline{a}\langle P\rangle$ mis en parallèle d'une réception de message $a(x).Q$.

La syntaxe de HOcore est la suivante.

$$P ::= a(x).P \;\big|\; \overline{a}\langle P\rangle \;\big|\; P \parallel P \;\big|\; x \;\big|\; \mathbf{0}$$

Un processus P peut soit être une réception de message sur un certain canal, notée $a(x).P$, soit une émission de message, notée $\overline{a}\langle P\rangle$, soit la mise en parallèle de processus $P \parallel Q$, soit une variable x, soit le processus inactif $\mathbf{0}$. Nous distinguons les *variables* x, y, z des *noms de canaux* a, b, c.

Intuitivement, l'unique règle de réduction est la règle de communication suivante, où l'opération $[P \,/\, x]Q$ est la *substitution* de la variable x par le processus P dans Q. Le processus P, émis sur le canal a, est transmis au préfixe de réception $a(x).Q$.

$$\overline{a}\langle P\rangle \parallel a(x).Q \longrightarrow [P \,/\, x]Q \qquad (\dagger)$$

Notons que le calcul est asynchrone : l'émission de message n'a pas de continuation. Dans un calcul synchrone, l'envoi de message est de la forme $\overline{a}\langle P\rangle.Q$, où le processus Q est la continuation démarrée après l'émission du message sur a. Nous montrerons dans la section 4.3 que les messages synchrones permettent un encodage de la KAM ne nécessitant qu'un seul nom de canal.

Variables Une variable x est dite *liée* si elle est sous la portée d'un lieur pour cette variable (une réception de message $a(x).P$), *libre* sinon. Par exemple, dans le processus $a(x).(P \parallel y)$ avec $x \neq y$, les occurrences de x dans P sont liées, mais y est libre.

La machine de Krivine pour le λ-calcul en appel par nom [4] permet de ne considérer que des termes clos, et d'éviter les difficultés usuelles dues à la capture de variables libres. Il en est de même avec l'encodage que nous présentons ici : nous ne manipulerons que des termes dont toutes les variables seront liées. Les noms de canaux sont eux tous libres, HOcore n'ayant pas d'opérateur de restriction.

2.2. Sémantique

La sémantique de HOcore est définie par un système de transitions étiquetées (LTS). Les étiquettes sont définies par la syntaxe suivante :

$$\alpha ::= \overline{a}\langle P\rangle \mid a(P) \mid \tau \;\;.$$

Elles correspondent respectivement à l'émission d'un processus, la réception d'un processus, et à une communication interne. Le LTS est défini inductivement, par les règles suivantes.

$$\frac{}{\overline{a}\langle P\rangle \xrightarrow{\overline{a}\langle P\rangle} \mathbf{0}}\ \textsc{Out} \qquad \frac{}{a(x).Q \xrightarrow{a(P)} [P\,/\,x]Q}\ \textsc{Inp}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q}\ \textsc{Par1} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'}\ \textsc{Par2}$$

$$\frac{P \xrightarrow{\overline{a}\langle R\rangle} P' \qquad Q \xrightarrow{a(R)} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}\ \textsc{Tau1} \qquad \frac{P \xrightarrow{a(R)} P' \qquad Q \xrightarrow{\overline{a}\langle R\rangle} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}\ \textsc{Tau2}$$

Congruence structurelle Notons que le LTS précédent est très rigide : il conserve strictement la structure des termes (à l'inverse, une sémantique réductionelle utilise généralement une notion de congruence structurelle, qui permet par exemple de réorganiser les parenthèses modulo associativité de la mise en parallèle). L'avantage est que les termes obtenus après une réduction sont plus facile à analyser ; l'inconvénient est que ce LTS tend à générer de nombreuses occurrences du processus $\mathbf{0}$, qui sont inutiles. Par exemple, on a formellement les transitions suivantes :

$$(\overline{a}\langle P\rangle \parallel \overline{b}\langle Q\rangle) \parallel a(x).b(y).(x \parallel y) \xrightarrow{\tau} (\mathbf{0} \parallel \overline{b}\langle Q\rangle) \parallel b(y).(P \parallel y) \xrightarrow{\tau} (\mathbf{0} \parallel \mathbf{0}) \parallel (P \parallel Q)$$

On va donc s'appuyer sur une notion de congruence structurelle très restreinte qui permet de s'affranchir de ces occurrences de $\mathbf{0}$.

Une relation $\mathcal{R}$ est une *congruence* si c'est une relation d'équivalence (réflexive, symétrique, transitive) qui respecte les différents constructeurs du langage (par exemple si $P\ \mathcal{R}\ Q$ alors nous avons $a(x).P\ \mathcal{R}\ a(x).Q$). On quotiente dans la suite l'ensemble des processus par la plus petite congruence telle que la composition parallèle admette le processus $\mathbf{0}$ comme élément neutre à gauche et à droite.

2.3. Expressivité

HOcore est qualifié de minimal, car il ne contient que le strict nécessaire à l'ordre supérieur. Par exemple, il n'inclut pas d'opérateurs de restriction ou de réplication. HOcore est tout de même Turing complet : un encodage fidèle des machines de Minsky est présenté dans [5]. En particulier, le problème de la terminaison y est indécidable.

Un objectif de ce papier est de montrer qu'il est possible d'encoder directement des modèles de calcul plus complexes, comme le λ-calcul, dans HOcore.

2.4. Équivalence de processus

Une des questions cruciales de l'étude de calculs de processus est de savoir si deux processus « font la même chose ». Ainsi, dans l'optique de la programmation modulaire, on doit être capable de dire si deux bibliothèques logicielles sont interchangeables. Deux processus sont équivalents si, dans n'importe quel contexte, ce que l'on observe de leur activité est similaire. Formellement, on définit la *congruence barbue* [6] comme la plus grande relation symétrique telle que :

— si $P \simeq Q$ et $P \xrightarrow{\tau} P'$, alors il existe un Q' tel que $Q \xrightarrow{\tau} Q'$ et $P' \simeq Q'$: la congruence barbue est préservée par réductions ;

— si $P \simeq Q$, alors $C[P] \simeq C[Q]$ pour tout contexte C, un contexte étant un processus avec un trou : la congruence barbue est une congruence ;

— si $P \simeq Q$ et $P \xrightarrow{\overline{a}\langle P''\rangle} P'$, alors il existe Q' et Q'' tels que $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$: la congruence barbue met en relation des processus avec les mêmes *observables*, ou barbes, qui sont ici la possibilité d'émettre un message sur un canal donné.

On peut définir de façon similaire la *congruence barbue faible*, notée $\approxeq$, en considérant dans la premiere clause des séquences arbitraires de transitions, plutôt que des transitions simples.

Un des résultats fondamentaux de HOcore est la décidabilité de la congruence barbue [5]. Les processus de HOcore ne pouvant pas cacher leurs réductions (le langage n'a pas d'opérateur de restriction), il est toujours possible de définir des contextes explorant la structure d'un processus. En revanche, dès que l'on autorise des réductions anonymes, par exemple grâce à des restrictions globales empêchant l'observation sur certains noms, la congruence barbue devient indécidable. Les travaux précédents ont montré que quatre telles restrictions globales suffisent.

3. La KAM

La machine abstraite de Krivine est une machine très simple pour évaluer les termes du λ-calcul en appel par nom. Elle permet également de définir des opérateurs de contrôle.

Une configuration de la KAM est composé d'un terme du λ-calcul et d'une pile, qui est une liste de λ-termes. Comme indiqué plus haut, tous les λ-termes considérés sont *clos* (ils ne contiennent pas de variable libre).

$$C ::= M \star \pi$$
$$M ::= x \mid MN \mid \lambda x.M$$
$$\pi ::= M :: \pi \mid []$$

Les règles de réductions de la KAM (sans opérateur de contrôle) sont les suivantes ; un calcul s'arrête quand un état $\lambda x.M \star []$ est atteint.

$$MN \star \pi \mapsto M \star N :: \pi \qquad \text{(Push)}$$
$$\lambda x.M \star N :: \pi \mapsto [N \,/\, x]M \star \pi \qquad \text{(Grab)}$$

La KAM avec opérateurs de contrôles ajoute deux constructions syntaxiques : l'opérateur de capture de continuation `cc`, utilisable dans les programmes, et les constantes de pile k_π, qui ne peuvent être créées que par appel à `cc`. Les deux règles suivantes sont alors ajoutées.

$$\texttt{cc} \star M :: \pi \mapsto M \star k_\pi :: \pi \qquad \text{(CallCC)}$$
$$k_\pi \star M :: \pi' \mapsto M \star \pi \qquad \text{(Restore)}$$

4. KAM en HOcore

Nous commençons par traduire la KAM sans opérateurs de contrôle ; nous étendons ensuite notre encodage à ces opérateurs en section 4.2.

4.1. Version asynchrone

Nous présentons un codage n'utilisant que deux noms libres. La pile courante est un message sur le nom c (“c” pour *continuation*). Le contenu de la pile est un message sur le nom a (“a” comme

argument), correspondant à la tête de la pile, et un message sur c contenant la queue de la pile. La traduction de la pile vide est arbitrairement fixée au processus $\overline{b}\langle\mathbf{0}\rangle$, pour un troisième nom libre b permettant d'observer la fin du calcul (les résultats démontrés ci-dessous sont toujours valides lorsque ce processus est remplacé par un processus arbitraire, tant que celui-ci n'introduit pas de divergence ou de non-déterminisme).

La traduction d'une pile $1 :: 2 :: 3 :: []$ prend donc la forme $\overline{a}\langle 1\rangle \parallel \overline{c}\langle\overline{a}\langle 2\rangle \parallel \overline{c}\langle\overline{a}\langle 3\rangle \parallel \overline{c}\langle\overline{b}\langle\mathbf{0}\rangle\rangle\rangle\rangle$.

Pour ne pas alourdir les notations, nous utilisons le même symbole pour traduire des états, des piles et des λ-termes. Nous supposons également que les noms des variables liées u et s (u n'apparaît que pour la traduction des opérateurs de contrôle) sont différents des noms de variables du λ-terme.

$$
\begin{aligned}
[\![[]]\!] &\triangleq \overline{b}\langle\mathbf{0}\rangle \\
[\![M :: \pi]\!] &\triangleq \overline{a}\langle[\![M]\!]\rangle \parallel \overline{c}\langle[\![\pi]\!]\rangle \\
[\![MN]\!] &\triangleq c(s).([\![M]\!] \parallel \overline{c}\langle\overline{a}\langle[\![N]\!]\rangle \parallel \overline{c}\langle s\rangle\rangle) \\
[\![\lambda x.M]\!] &\triangleq c(s).(a(x).\,[\![M]\!] \parallel s) \\
[\![x]\!] &\triangleq x \\
[\![M \star \pi]\!] &\triangleq [\![M]\!] \parallel \overline{c}\langle[\![\pi]\!]\rangle
\end{aligned}
$$

Notons que dans la traduction d'une configuration, la pile, composée de messages imbriqués sur a et c, est elle même encapsulée à l'intérieur d'un message sur le nom c.

Nous montrons ci-dessous que la réduction de $[\![M \star \pi]\!]$ est déterministe pour tout M et tout π. De plus, à chaque étape de calcul de la KAM correspond une ou deux étapes de calcul du processus traduit (ou trois lorsque l'on prend en compte les opérateurs de contrôle).

Lemme 1 (Substitution)**.** *Pour tous M, x, N avec N clos, nous avons $[\![[N \,/\, x]M]\!] = [[\![N]\!] \,/\, x]\,[\![M]\!]$.*

Démonstration. Par une simple induction sur M. □

Lemme 2 (Simulation)**.** *Pour tous M, M', π, π' tels que $M \star \pi \mapsto M' \star \pi'$, nous avons $[\![M \star \pi]\!] \xrightarrow{\tau}{}^{+} [\![M' \star \pi']\!]$.*

Démonstration. Il suffit de considérer les règles de réduction de la machine de Krivine :

Push ($MN \star \pi \mapsto M \star N :: \pi$) : on vérifie que

$$
\begin{aligned}
[\![MN \star \pi]\!] &= (c(s).\,[\![M]\!] \parallel \overline{c}\langle\overline{a}\langle[\![N]\!]\rangle \parallel \overline{c}\langle s\rangle\rangle) \parallel \overline{c}\langle[\![\pi]\!]\rangle \\
&\xrightarrow{\tau} [\![M]\!] \parallel \overline{c}\langle\overline{a}\langle[\![N]\!]\rangle \parallel \overline{c}\langle[\![\pi]\!]\rangle\rangle \\
&= [\![M \star N :: \pi]\!] \quad .
\end{aligned}
$$

Il suffit donc d'une seule transition pour simuler cette règle.

Grab ($\lambda x.M \star N :: \pi \mapsto [N \,/\, x]M \star \pi$) : il faut cette fois deux transitions :

$$
\begin{aligned}
[\![\lambda x.M \star N :: \pi]\!] &= (c(s).(a(x).\,[\![M]\!]) \parallel s) \parallel \overline{c}\langle\overline{a}\langle[\![N]\!]\rangle \parallel \overline{c}\langle[\![\pi]\!]\rangle\rangle \\
&\xrightarrow{\tau} (a(x).\,[\![M]\!]) \parallel \overline{a}\langle[\![N]\!]\rangle \parallel \overline{c}\langle[\![\pi]\!]\rangle \\
&\xrightarrow{\tau} [[\![N]\!] \,/\, x]\,[\![M]\!] \parallel \overline{c}\langle[\![\pi]\!]\rangle \\
&= [\![[N \,/\, x]M]\!] \parallel \overline{c}\langle[\![\pi]\!]\rangle \qquad \text{(par le Lemme 1)} \\
&= [\![[N \,/\, x]M \star \pi]\!] \quad .
\end{aligned}
$$

□

Il nous faut maintenant prouver que les transitions des processus traduits sont déterministes, et qu'elles correspondent à des réductions dans la KAM. Or, comme on le voit dans la preuve précédente, certaines transitions sont intermédiaires et doivent être complétées afin de correspondre précisément à une réduction de la KAM.

Afin de simplifier la preuve, nous passons par une machine abstraite légèrement différente de la KAM, dans laquelle certaines étapes sont artificiellement dédoublées : les étapes de calcul des processus traduits correspondent ainsi exactement aux réductions de la KAM modifiée.

La modification est la suivante : on introduit une configuration intermédiaire, notée $\lambda' x.M \star \pi$, et la règle Grab est dédoublée comme suit :

$$\lambda x.M \star \pi \mapsto \lambda' x.M \star \pi \qquad (\textsc{Grab}_1)$$
$$\lambda' x.M \star N :: \pi \mapsto [N \,/\, x]M \star \pi \qquad (\textsc{Grab}_2)$$

Fait 3. *Une configuration admet une séquence infinie de réductions dans la KAM originelle si et seulement elle admet une séquence infinie de réductions dans la KAM modifiée.*

En accord avec le second cas dans la preuve du lemme 2, la fonction de traduction est étendue aux configurations intermédiaires en posant :

$$\left[\!\left[\lambda' x.M \star \pi\right]\!\right] \stackrel{\triangle}{=} (a(x).\,[\![M]\!]) \parallel [\![\pi]\!] \quad .$$

Lemme 4. *La fonction de traduction est injective.*

Démonstration. On prouve qu'elle est injective sur les λ-termes, puis sur les piles, puis sur les configurations. □

Notons $[\![\cdot]\!]^{-1}$ la fonction partielle inverse de la traduction, i.e., telle que $[\![[\![C]\!]]\!]^{-1} = C$ pour toute configuration C de la KAM modifiée. On se convainc aisément que cette fonction est calculable.

Lemme 5 (Réflection)**.** *Pour toute configuration C et processus P, si $[\![C]\!] \xrightarrow{\tau} P$, alors $[\![P]\!]^{-1}$ est défini et $C \mapsto [\![P]\!]^{-1}$.*

Démonstration. On raisonne par cas sur la configuration C :

- $C = MN \star \pi$: on a $[\![C]\!] = (c(s).\,[\![M]\!] \parallel \overline{c}\langle \overline{a}\langle [\![N]\!]\rangle \parallel \overline{c}\langle s\rangle\rangle) \parallel \overline{c}\langle [\![\pi]\!]\rangle$, d'où $P = [\![M]\!] \parallel \overline{c}\langle \overline{a}\langle [\![N]\!]\rangle \parallel \overline{c}\langle [\![\pi]\!]\rangle\rangle$ puisqu'une seule transition est possible. On vérifie alors que $[\![P]\!]^{-1} = M \star N :: \pi$, et $C \mapsto [\![P]\!]^{-1}$ par la règle (Push).
- $C = \lambda x.MN \star \pi$: on a $[\![C]\!] = (c(s).(a(x).\,[\![M]\!]) \parallel s) \parallel \overline{c}\langle [\![\pi]\!]\rangle$, d'où $P = (a(x).\,[\![M]\!]) \parallel [\![\pi]\!]$. On vérifie alors que $[\![P]\!]^{-1} = \lambda' x.M \star \pi$, et $C \mapsto [\![P]\!]^{-1}$ par la règle ($\textsc{Grab}_1$).
- $C = \lambda' x.MN \star \pi$: si la pile π est vide, alors $[\![C]\!] = (a(x).\,[\![M]\!]) \parallel \overline{b}\langle \mathbf{0}\rangle$ n'admet pas de transition τ, ce qui contredit l'hypothèse sur P. On a donc $\pi = N :: \pi'$, et $[\![C]\!] = (a(x).\,[\![M]\!]) \parallel \overline{a}\langle N\rangle \parallel \overline{c}\langle [\![\pi']\!]\rangle$. On a nécessairement $P = [[\![N]\!] \,/\, x]\,[\![M]\!] \parallel \overline{c}\langle [\![\pi']\!]\rangle$; on vérifie alors que $[\![P]\!]^{-1} = [N \,/\, x]M \star \pi'$ à l'aide du Lemme 1, et que $C \mapsto [\![P]\!]^{-1}$ par la règle ($\textsc{Grab}_2$). □

Théorème 6 (Déterminisme)**.** *Pour toute configuration C et tous processus P, P', P'', si $[\![C]\!] \xrightarrow{\tau}{}^{\star} P$, $P \xrightarrow{\tau} P'$ et $P \xrightarrow{\tau} P''$, alors $P' = P''$.*

Démonstration. Par le Lemme 5, on peut se ramener au cas où $P = [\![C]\!]$. En reprenant la preuve de ce même lemme, par analyse de cas sur C, on constate qu'au plus une communication est possible dans le processus traduit $[\![C]\!]$. □

Théorème 7 (Correspondance opérationelle)**.** *Pour toutes configurations C, C', $C \mapsto C'$ si et seulement si $[\![C]\!] \xrightarrow{\tau} [\![C']\!]$.*

Démonstration. Conséquence immédiate des Lemmes 2 et 5. □

Théorème 8. *Pour toute configuration C, nous avons C termine si et seulement si $[\![C]\!] \xrightarrow{\tau}{}^{\star} \xrightarrow{\overline{b}\langle \mathbf{0} \rangle}$.*

Démonstration. La machine abstraite s'arrête exactement sur les configurations de la forme $\lambda' x.M \star []$, dont les encodages sont de la forme $(a(x).[\![M]\!]) \parallel \overline{b}\langle \mathbf{0} \rangle$, qui ont pas de transitions τ, et qui ont une barbe sur b. Inversement, les seules configurations dont la traduction est capable d'émettre sur b sont celles de la forme $\lambda' x.M \star []$. On peut donc conclure par le Théorème 7. □

En tant que fragment du π-calcul d'ordre supérieur [7], HOcore peut naturellement être étendu en ajoutant un opérateur de restriction de nom. Nous ne considérons ici qu'une extension plus réduite n'utilisant que des restrictions globales (i.e., les restrictions ne peuvent être dans des messages, donc elles ne peuvent être répliquées). La syntaxe est étendue de la manière suivante.

$$T ::= \nu a.T \mid P$$

L'extension du LTS est immédiate : les seules transitions autorisées pour les termes $\nu a.T$ sont celles de T dont les noms ne mentionnent pas a. Nous notons $\mathtt{n}(\alpha)$ les noms de canaux de α.

$$\frac{T \xrightarrow{\alpha} T' \qquad a \notin \mathtt{n}(\alpha)}{\nu a.T \xrightarrow{\alpha} \nu a.T'}$$

Intuitivement, $\nu a.P$ correspond au processus P auquel on interdit de communiquer sur a avec l'extérieur (en émission comme en réception) : le nom de canal a est connu de lui seul, toutes les communications sur a ne peuvent donc avoir lieu qu'à l'intérieur de P.

Théorème 9. *Soit $\Omega \triangleq \nu a.\overline{a}\langle a(x).(\overline{a}\langle x \rangle \parallel x) \rangle \parallel a(x).(\overline{a}\langle x \rangle \parallel x)$. Pour tout λ-terme M et toute pile π, on a $\nu a.\nu c.[\![M \star \pi]\!] \simeq \Omega$ si et seulement si $M \star \pi$ ne termine pas.*

Démonstration. Ω est un processus divergent, dont la seule transition est $\Omega \xrightarrow{\tau} \Omega$. Par conséquent, si $M \star \pi$ ne termine pas, alors $\nu a.\nu c.[\![M \star \pi]\!]$ lui est équivalent, puisqu'il ne pourra jamais émettre sur son seul nom libre (b). Inversement, si $M \star \pi$ termine, alors $\nu a.\nu c.[\![M \star \pi]\!]$ finira par émettre sur b, ce qui le distingue du processus Ω. □

Corollaire 10. *L'équivalence contextuelle est indécidable dans HOcore avec deux restrictions globales.*

Un raisonnement similaire permet d'obtenir l'indécidabilité de l'équivalence contextuelle faible dans HOcore avec deux restrictions globales ; on peut même utiliser dans ce cas le processus vide, $\mathbf{0}$, plutôt que le processus divergent Ω (notons cependant que pour cette preuve, on n'a pas la possibilité d'encoder la pile vide par un processus arbitraire ; en particulier, le processus vide ne conviendrait pas : il est nécessaire d'avoir une observable visible lorsque le fond de pile est atteint). En contrepartie, dans le cas fort, le fond de pile peut être traduit par $\mathbf{0}$ en effectuant la comparaison avec Ω.

4.2. Opérateurs de contrôle

Nous ajoutons maintenant les opérateurs de contrôle (call-cc). Rappelons les deux règles de réduction de la KAM définissant ces opérateurs :

$$\mathtt{cc} \star M :: \pi \mapsto M \star k_\pi :: \pi \qquad \text{(CallCC)}$$
$$k_\pi \star M :: \pi' \mapsto M \star \pi \qquad \text{(Restore)}$$

Etant donné un processus P, on définit

$$K(P) \triangleq c(s_0).(s_0 \parallel a(u).c(\_).(u \parallel \overline{c}\langle P\rangle)).$$

Les deux opérateurs sont alors traduits comme suit :

$$\begin{aligned}[\![\mathtt{cc}]\!] &\triangleq c(s_0).(s_0 \parallel c(s).a(u).(u \parallel \overline{c}\langle\overline{a}\langle K(s)\rangle \parallel \overline{c}\langle s\rangle\rangle))\\ [\![k_\pi]\!] &\triangleq K([\![\pi]\!])\end{aligned}$$

Cet encodage nécessite trois transitions pour simuler les règles (CallCC) et (Restore) de la KAM avec opérateurs de contrôle. Comme précédemment pour la règle (Grab), on introduit donc quatre configurations intermédiaires et artificielles dans la KAM, dont la sémantique est définie par les six règles suivantes.

$$\mathtt{cc} \star \pi \mapsto \mathtt{cc}_1 \star \pi \qquad (\text{CallCC}_1)$$
$$\mathtt{cc}_1 \star M :: \pi \mapsto \mathtt{cc}_2 \star M :: \pi \qquad (\text{CallCC}_2)$$
$$\mathtt{cc}_2 \star M :: \pi \mapsto M \star k_\pi :: \pi \qquad (\text{CallCC}_3)$$

$$k_\pi \star \pi' \mapsto k^1_\pi \star \pi' \qquad (\text{Restore}_1)$$
$$k^1_\pi \star M :: \pi' \mapsto k^2_\pi \star M :: \pi' \qquad (\text{Restore}_2)$$
$$k^2_\pi \star M :: \pi' \mapsto M \star \pi \qquad (\text{Restore}_3)$$

Ces quatre configurations s'encodent comme suit dans HOcore.

$$\begin{aligned}[\![\mathtt{cc}_1 \star \pi]\!] &\triangleq [\![\pi]\!] \parallel c(s).a(u).(u \parallel \overline{c}\langle\overline{a}\langle K(s)\rangle \parallel \overline{c}\langle s\rangle\rangle)\\ [\![\mathtt{cc}_2 \star M :: \pi]\!] &\triangleq \overline{a}\langle[\![M]\!]\rangle \parallel a(u).(u \parallel \overline{c}\langle\overline{a}\langle K([\![\pi]\!])\rangle \parallel \overline{c}\langle[\![\pi]\!]\rangle\rangle)\\ &= \overline{a}\langle[\![M]\!]\rangle \parallel a(u).(u \parallel \overline{c}\langle\overline{a}\langle[\![k_\pi]\!]\rangle \parallel \overline{c}\langle[\![\pi]\!]\rangle\rangle)\\ &= \overline{a}\langle[\![M]\!]\rangle \parallel a(u).(u \parallel \overline{c}\langle[\![k_\pi :: \pi]\!]\rangle)\\ [\![k^1_\pi \star \pi']\!] &\triangleq [\![\pi']\!] \parallel a(u).c(\_).(u \parallel \overline{c}\langle[\![\pi]\!]\rangle)\\ [\![k^2_\pi \star M :: \pi']\!] &\triangleq \overline{c}\langle[\![\pi']\!]\rangle \parallel c(\_).([\![M]\!] \parallel \overline{c}\langle[\![\pi]\!]\rangle)\\ &= \overline{c}\langle[\![\pi']\!]\rangle \parallel c(\_).\,[\![M \star \pi]\!]\end{aligned}$$

On vérifie alors que les processus traduits correspondants ont exactement les transitions suivantes.

$$\begin{aligned}&[\![\mathtt{cc} \star M :: \pi]\!] \xrightarrow{\tau} [\![\mathtt{cc}_1 \star M :: \pi]\!] \xrightarrow{\tau} [\![\mathtt{cc}_2 \star M :: \pi]\!] \xrightarrow{\tau} [\![M \star k_\pi :: \pi]\!]\\ &[\![k_\pi \star M :: \pi']\!] \xrightarrow{\tau} [\![k^1_\pi \star M :: \pi']\!] \xrightarrow{\tau} [\![k^2_\pi \star M :: \pi']\!] \xrightarrow{\tau} [\![M \star \pi]\!]\end{aligned}$$

Les preuves des lemmes 1, 2, 4 et 5 ainsi que des théorèmes 6 et 7 s'étendent sans difficulté.

Le lecteur attentif aura remarqué que la traduction de l'opérateur cc effectue d'abord une réception $c(s)$ avant la réception $a(u)$. Ce choix est purement esthétique : changer l'ordre des réceptions est tout à fait possible, mais ne permet pas d'utiliser $[\![k_\pi :: \pi]\!]$ dans la traduction de $\mathtt{cc}_2$, la rendant moins succincte.

4.3. Version synchrone

Nous montrons maintenant qu'il est possible d'obtenir une traduction de la KAM en utilisant un seul nom, si le calcul considéré est *synchrone*. Une version synchrone de HOcore ne modifie que

l'émission de message, remplaçant la construction $\overline{a}\langle P\rangle$ par $\overline{a}\langle P\rangle.Q$. Cette dernière émet toujours un message sur a transportant P. En revanche, dès que le message est reçu, le processus Q, appelé *continuation*, est lancé : la règle de réduction intuitive devient

$$\overline{a}\langle P\rangle.Q \parallel a(x).R \longrightarrow Q \parallel [P \,/\, x]R \qquad (\ddagger)$$

Formellement, cela se traduit dans le LTS par une simple modification de la règle axiome Out :

$$\frac{}{\overline{a}\langle P\rangle.Q \xrightarrow{\overline{a}\langle P\rangle} Q}\ \textsc{Out}$$

La traduction de la KAM (étendue) est donnée ci-dessous. Nous traduisons désormais les piles comme étant des messages synchrones : le contenu du message étant la tête de la pile et la continuation du message la queue de la pile. Nous traduisons ainsi une pile $1 :: 2 :: 3 :: []$ par $\overline{a}\langle 1\rangle.\overline{a}\langle\overline{a}\langle 2\rangle.\overline{a}\langle\overline{a}\langle 3\rangle.\overline{a}\langle\overline{b}\langle\mathbf{0}\rangle\rangle\rangle\rangle$.

Au sein d'une configuration, comme dans le cas asynchrone, la pile sera de plus encapsulée à l'intérieur d'un message additionel sur a.

Comme ci-dessus, pour tout processus P, nous définissons

$$K(P) \triangleq a(s_0).(s_0 \parallel a(u).a(\_).(u \parallel \overline{a}\langle P\rangle)).$$

(Par convention, nous notons $\overline{a}\langle P\rangle$ les messages $\overline{a}\langle P\rangle.\mathbf{0}$ dont la continuation est vide). Nous définissons alors la traduction synchrone comme suit.

$$\begin{aligned}
[\![[]]\!] &\triangleq \overline{b}\langle\mathbf{0}\rangle \\
[\![M :: \pi]\!] &\triangleq \overline{a}\langle[\![M]\!]\rangle.\overline{a}\langle[\![\pi]\!]\rangle \\
[\![MN]\!] &\triangleq a(s).([\![M]\!] \parallel \overline{a}\langle\overline{a}\langle[\![N]\!]\rangle.\overline{a}\langle s\rangle\rangle) \\
[\![\lambda x.M]\!] &\triangleq a(s).(a(x).\,[\![M]\!] \parallel s) \\
[\![x]\!] &\triangleq x \\
[\![M \star \pi]\!] &\triangleq [\![M]\!] \parallel \overline{a}\langle[\![\pi]\!]\rangle \\
[\![\mathtt{cc}]\!] &\triangleq a(s_0).(s_0 \parallel a(u).a(s).(u \parallel \overline{a}\langle\overline{a}\langle K(s)\rangle.\overline{a}\langle s\rangle\rangle)) \\
[\![k_\pi]\!] &\triangleq K([\![\pi]\!])
\end{aligned}$$

A nouveau, les preuves de la section 4.1 s'adaptent sans difficulté. L'équivalence contextuelle est donc indécidable dans HOcore synchrone avec une seule restriction de nom globale.

5. Conclusion

Nous avons présenté deux traductions directes de la machine abstraite de Krivine avec opérateurs de contrôle dans HOcore, utilisant un nom libre dans le cas synchrone, et deux dans le cas asynchrone. Cela nous a permis d'affiner la borne sur le nombre de restrictions globales de noms de canaux suffisant pour obtenir l'indécidabilité des équivalences contextuelles fortes et faible (en l'absence de restriction de nom, l'équivalence contextuelle forte est décidable que le calcul soit synchrone ou asynchrone [5]—le cas faible est ouvert).

Nous pouvons remarquer que les trois fonctionnalités fondamentales utilisées pour traduire la KAM sont les suivantes. Les deux premières portent sur l'ordre supérieur : les messages transportent des processus, et la réception effectue une substitution identique à celle du λ-calcul. La troisième fonctionnalité sur laquelle nous nous appuyons porte sur le contrôle de l'évaluation : nous devons

distinguer entre la tête de la pile et le reste de la pile. Pour ce faire, nous pouvons utiliser deux noms différents (version asynchrone), ou séquentialiser les émissions (version synchrone) et n'utiliser ainsi qu'un seul nom. Nous ne pensons pas qu'il soit possible d'obtenir une traduction de la KAM avec un seul nom dans un calcul asynchrone.

La relative simplicité de notre traduction, et le fait qu'elle permette de traiter immédiatement les opérateurs de contrôle, nous laisse espérer qu'elle permettra une meilleure compréhension de l'expressivité du calcul HOcore, et peut-être, à terme, de résoudre la question de la décidabilité de l'équivalence contextuelle faible—dans le calcul sans restriction de nom.

Références

[1] J. Aranda, F. D. Valencia, and C. Versari. On the expressive power of restriction and priorities in ccs with replication. In L. Alfaro, editor, *Foundations of Software Science and Computational Structures*, volume 5504 of *Lecture Notes in Computer Science*, pages 242–256. Springer Berlin Heidelberg, 2009.

[2] S. Boulier and A. Schmitt. Formalisation de hocore en coq. In *Actes des 23èmes Journées Francophones des Langages Applicatifs*, Jan. 2012.

[3] N. Busi, M. Gabbrielli, and G. Zavattaro. On the expressive power of recursion, replication and iteration in process calculi. *Mathematical Structures in Computer Science*, 19(6) :1191–1222, Dec. 2009.

[4] J.-L. Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3) :199–207, 2007.

[5] I. Lanese, J. A. Pérez, D. Sangiorgi, and A. Schmitt. On the expressiveness and decidability of higher-order process calculi. *Information and Computation*, 209(2) :198–226, Feb. 2011. Extended abstract presented at *Logic in Computer Science (LICS)*, 2008.

[6] R. Milner and D. Sangiorgi. Barbed bisimulation. In *19th ICALP*, volume 623 of *LNCS*, pages 685–695. Springer Verlag, 1992.

[7] D. Sangiorgi and D. Walker. *The π-calculus : a Theory of Mobile Processes*. Cambridge University Press, 2001.